autogluon_each_location

October 7, 2023

[50]:
```
# config
run_analysis = False
```

[51]:
```
import pandas as pd
import numpy as np




import warnings
warnings.filterwarnings("ignore")

def fix_datetime(X, name):
    # Convert 'date_forecast' to datetime format and replace original column␣
 ↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)

    # Drop rows where the minute part of the time is not 0
    X = X[X.index.minute == 0]
    return X



def convert_to_datetime(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_datetime(X_train_observed, "X_train_observed")
    X_train_estimated = fix_datetime(X_train_estimated, "X_train_estimated")
    X_test = fix_datetime(X_test, "X_test")

    # add sample weights, which are 1 for observed and 3 for estimated
    X_train_observed["sample_weight"] = 1
    X_train_estimated["sample_weight"] = 3
    X_test["sample_weight"] = 3

    X_train_observed["estimated_diff_hours"] = 0
    X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index - pd.
 ↪to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
    X_test["estimated_diff_hours"] = (X_test.index - pd.
 ↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

    X_train_estimated["estimated_diff_hours"] =␣
 ↪X_train_estimated["estimated_diff_hours"].astype('int64')
    # the filled once will get dropped later anyways, when we drop y nans
    X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].fillna(-50).
 ↪astype('int64')


    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)

    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train




def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,␣
 ↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =␣
 ↪convert_to_datetime(X_train_observed, X_train_estimated, X_test, y_train)

    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])

    # fill missng sample_weight with 3
    #X_train["sample_weight"] = X_train["sample_weight"].fillna(0)


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

    X_train = pd.merge(X_train, y_train, how="inner", left_index=True,␣
 ↪right_index=True)

    # print number of nans in sample_weight
```

```python
    print(f"Number of nans in sample_weight: {X_train['sample_weight'].isna().
↪sum()}")
    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
Number of nans in sample_weight: 0
Number of nans in y: 0
Processing location B…
Number of nans in sample_weight: 0
Number of nans in y: 4
```

```
Processing location C…
Number of nans in sample_weight: 0
Number of nans in y: 6059
```

1 Feature enginering

[52]:
```
# temporary
X_train["hour"] = X_train.index.hour
X_train["weekday"] = X_train.index.weekday
# weekday or is_weekend
X_train["is_weekend"] = X_train["weekday"].apply(lambda x: 1 if x >= 5 else 0)

# drop weekday
#X_train.drop(columns=["weekday"], inplace=True)
X_train["month"] = X_train.index.month
X_train["year"] = X_train.index.year

X_test["hour"] = X_test.index.hour
X_test["weekday"] = X_test.index.weekday

# weekday or is_weekend
X_test["is_weekend"] = X_test["weekday"].apply(lambda x: 1 if x >= 5 else 0)

# drop weekday
#X_test.drop(columns=["weekday"], inplace=True)
X_test["month"] = X_test.index.month
X_test["year"] = X_test.index.year



to_drop = ["snow_drift:idx", "snow_density:kgm3"]

X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.dropna(subset=['y'], inplace=True)
X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

[53]:
```
import autogluon.eda.auto as auto

if run_analysis:
    auto.dataset_overview(train_data=X_train, test_data=X_test, label="y",␣
 ↪sample=None)
```

[54]:
```
if run_analysis:
    auto.target_analysis(train_data=X_train, label="y")
```

2 Starting

[55]:
```
import os


# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
 ↪filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 78
Now creating submission number: 79
New filename: submission_79
```

[56]:
```
from autogluon.tabular import TabularDataset, TabularPredictor
train_data = TabularDataset('X_train_raw.csv')
train_data.drop(columns=['ds'], inplace=True)

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60
presets = 'best_quality'

sample_weight = 'sample_weight' #None
weight_evaluation = True #False
```

```
Loaded data from: X_train_raw.csv | Columns = 53 / 53 | Rows = 92951 -> 92951
```

[57]:
```
predictors = [None, None, None]
```

[58]:
```
loc = "A"
print(f"Training model for location {loc}...")
predictor = TabularPredictor(label=label, eval_metric=metric,␣
 ↪path=f"AutogluonModels/{new_filename}_{loc}", sample_weight=sample_weight,␣
 ↪weight_evaluation=weight_evaluation).fit(train_data[train_data["location"]␣
 ↪== loc], time_limit=time_limit, presets=presets)
predictors[0] = predictor
```

```
Warning: path already exists! This predictor may overwrite an existing
```

```
predictor! path="AutogluonModels/submission_79_A"
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8,
num_bag_sets=20
Values in column 'sample_weight' used as sample weights instead of predictive
features. Evaluation will report weighted metrics, so ensure same column exists
in test data.
Beginning AutoGluon training … Time limit = 60s
AutoGluon will save models to "AutogluonModels/submission_79_A/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.191-1 (2023-08-16)
Disk Space Avail:   102.11 GB / 105.09 GB (97.2%)
Train Data Rows:    34061
Train Data Columns: 51
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 631.01116,
1166.20607)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    31355.98 MB
        Train Data (Original)  Memory Usage: 15.33 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 4 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:

Training model for location A…

                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
```

```
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 42 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  6 | ['estimated_diff_hours', 'hour', 'weekday',
'is_weekend', 'month', …]
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])       :  5 | ['estimated_diff_hours', 'hour',
'weekday', 'month', 'year']
                ('int', ['bool']) :  4 | ['is_day:idx', 'is_in_shadow:idx',
'wind_speed_w_1000hPa:ms', 'is_weekend']
        0.2s = Fit runtime
        48 features in original data used to generate 48 features in processed
data.
        Train Data (Processed) Memory Usage: 12.13 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.27s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
AutoGluon will fit 2 stack levels (L1 to L2) …
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 39.81s of the
59.72s of remaining time.
        -277.2896        = Validation score   (-mean_absolute_error)
        0.05s    = Training   runtime
        1.65s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 38.01s of the
57.92s of remaining time.
        -278.2945        = Validation score   (-mean_absolute_error)
        0.05s    = Training   runtime
        1.65s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 36.18s of the
56.1s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -148.1788        = Validation score   (-mean_absolute_error)
        31.56s   = Training   runtime
        24.68s   = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 59.73s of the
17.21s of remaining time.
        -148.1788        = Validation score   (-mean_absolute_error)
        0.36s    = Training   runtime
        0.0s     = Validation runtime
Fitting 9 L2 models …
Fitting model: LightGBMXT_BAG_L2 … Training model for up to 16.84s of the
16.81s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -149.3082        = Validation score   (-mean_absolute_error)
        7.95s    = Training   runtime
        1.15s    = Validation runtime
Fitting model: LightGBM_BAG_L2 … Training model for up to 5.29s of the 5.27s
of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -147.0044        = Validation score   (-mean_absolute_error)
        3.9s     = Training   runtime
        0.37s    = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L3 … Training model for up to 59.73s of the
-2.34s of remaining time.
        -146.5428        = Validation score   (-mean_absolute_error)
        0.28s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 62.67s … Best model:
```

```
"WeightedEnsemble_L3"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_79_A/")
```

```
[ ]: loc = "B"
     print(f"Training model for location {loc}...")
     predictor = TabularPredictor(label=label, eval_metric=metric,␣
       ↪path=f"AutogluonModels/{new_filename}_{loc}", sample_weight=sample_weight,␣
       ↪weight_evaluation=weight_evaluation).fit(train_data[train_data["location"]␣
       ↪== loc], time_limit=time_limit, presets=presets)
     predictors[1] = predictor
```

```
Warning: path already exists! This predictor may overwrite an existing
predictor! path="AutogluonModels/submission_79_B"
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=1, num_bag_folds=8,
num_bag_sets=20
Values in column 'sample_weight' used as sample weights instead of predictive
features. Evaluation will report weighted metrics, so ensure same column exists
in test data.
Beginning AutoGluon training … Time limit = 60s
AutoGluon will save models to "AutogluonModels/submission_79_B/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.191-1 (2023-08-16)
Disk Space Avail:   102.08 GB / 105.09 GB (97.1%)
Train Data Rows:    32819
Train Data Columns: 51
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (1152.3, -0.0, 96.89334, 194.00409)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    30241.82 MB
        Train Data (Original)  Memory Usage: 14.77 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 4 features to boolean dtype as they
only contain 2 unique values.
```

```
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
Training model for location B…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 42 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  6 | ['estimated_diff_hours', 'hour', 'weekday',
'is_weekend', 'month', …]
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])       :  5 | ['estimated_diff_hours', 'hour',
'weekday', 'month', 'year']
                ('int', ['bool']) :  4 | ['is_day:idx', 'is_in_shadow:idx',
'wind_speed_w_1000hPa:ms', 'is_weekend']
        0.2s = Fit runtime
        48 features in original data used to generate 48 features in processed
data.
        Train Data (Processed) Memory Usage: 11.68 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.28s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
AutoGluon will fit 2 stack levels (L1 to L2) …
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 39.8s of the
59.71s of remaining time.
        -52.6414         = Validation score   (-mean_absolute_error)
        0.05s    = Training   runtime
        1.61s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 38.05s of the
57.96s of remaining time.
        -52.5565         = Validation score   (-mean_absolute_error)
        0.08s    = Training   runtime
        1.55s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 36.11s of the
56.02s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
```

[ ]:
```python
loc = "C"
print(f"Training model for location {loc}...")
predictor = TabularPredictor(label=label, eval_metric=metric,␣
 ↪path=f"AutogluonModels/{new_filename}_{loc}", sample_weight=sample_weight,␣
 ↪weight_evaluation=weight_evaluation).fit(train_data[train_data["location"]␣
 ↪== loc], time_limit=time_limit, presets=presets)
predictors[2] = predictor
```

3 Submit

[ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt

train_data_with_dates = TabularDataset('X_train_raw.csv')
train_data_with_dates["ds"] = pd.to_datetime(train_data_with_dates["ds"])

test_data = TabularDataset('X_test_raw.csv')
```

```
test_data["ds"] = pd.to_datetime(test_data["ds"])
#test_data
```

```
[ ]: test_ids = TabularDataset('test.csv')
     test_ids["time"] = pd.to_datetime(test_ids["time"])
     # merge test_data with test_ids
     test_data_merged = pd.merge(test_data, test_ids, how="inner", right_on=["time",␣
      ↪"location"], left_on=["ds", "location"])

     #test_data_merged
```

```
[ ]: # predict, grouped by location
     predictions = []
     location_map = {
         "A": 0,
         "B": 1,
         "C": 2
     }
     for loc, group in test_data.groupby('location'):
         i = location_map[loc]
         subset = test_data_merged[test_data_merged["location"] == loc].
      ↪reset_index(drop=True)
         #print(subset)
         pred = predictors[i].predict(subset)
         subset["prediction"] = pred
         predictions.append(subset)
```

```
[ ]: if run_analysis:
         # plot predictions for location A, in addition to train data for A
         for loc, idx in location_map.items():
             fig, ax = plt.subplots(figsize=(20, 10))
             # plot train data
             train_data_with_dates[train_data_with_dates["location"]==loc].
      ↪plot(x='ds', y='y', ax=ax, label="train data")

             # plot predictions
             predictions[idx].plot(x='ds', y='prediction', ax=ax,␣
      ↪label="predictions")

             # title
             ax.set_title(f"Predictions for location {loc}")
```

```
[ ]: # concatenate predictions
     submissions_df = pd.concat(predictions)
     submissions_df = submissions_df[["id", "prediction"]]
     submissions_df
```

```
[ ]: # Save the submission DataFrame to submissions folder, create new name based on␣
     ↪last submission, format is submission_<last_submission_number + 1>.csv

     # Save the submission
     print(f"Saving submission to submissions/{new_filename}.csv")
     submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),␣
     ↪index=False)
```

```
[ ]: # save this notebook to submissions folder
     import subprocess
     import os
     subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
     ↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
     ↪ipynb"])
```

```
[ ]: # feature importance
     location="A"
     split_time = pd.Timestamp("2022-10-28 22:00:00")
     estimated = train_data_with_dates[train_data_with_dates["ds"] >= split_time]
     estimated = estimated[estimated["location"] == location]
     # predictors[0].feature_importance(feature_stage="original", data=estimated,␣
     ↪time_limit=60*10)
```

```
[ ]: # feature importance
     observed = train_data_with_dates[train_data_with_dates["ds"] < split_time]
     observed = observed[observed["location"] == location]
     # predictor.feature_importance(feature_stage="original", data=observed,␣
     ↪time_limit=60*10)
```

```
[ ]: subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
     ↪join('notebook_pdfs', f"{new_filename}_with_feature_importance.pdf"),␣
     ↪"autogluon_each_location.ipynb"])
```

```
[ ]: import subprocess

     def execute_git_command(directory, command):
         """Execute a Git command in the specified directory."""
         try:
             result = subprocess.check_output(['git', '-C', directory] + command,␣
     ↪stderr=subprocess.STDOUT)
             return result.decode('utf-8').strip(), True
         except subprocess.CalledProcessError as e:
             print(f"Git command failed with message: {e.output.decode('utf-8').
     ↪strip()}")
             return e.output.decode('utf-8').strip(), False

     git_repo_path = "."
```

```
execute_git_command(git_repo_path, ['config', 'user.email', 'henrikskog01@gmail.
 ↪com'])
execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is␣
 ↪not None else 'Henrik eller Jørgen'])

branch_name = new_filename

# add datetime to branch name
branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

commit_msg = "run result"

execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# Navigate to your repo and commit changes
execute_git_command(git_repo_path, ['add', '.'])
execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# Push to remote
output, success = execute_git_command(git_repo_path, ['push',␣
 ↪'origin',branch_name])

# If the push fails, try setting an upstream branch and push again
if not success and 'upstream' in output:
    print("Attempting to set upstream and push again...")
    execute_git_command(git_repo_path, ['push', '--set-upstream',␣
 ↪'origin',branch_name])
    execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

execute_git_command(git_repo_path, ['checkout', 'main'])
```